Learning from Rational Behavior:
Predicting Solutions to Unknown Linear Programs

Shahin Jabbari, Ryan Rogers, Aaron Roth, Zhiwei Steven Wu
University of Pennsylvania
{jabbari@cis, ryrogers@sas, aaroth@cis, wuzhiwei@cis}.upenn.edu

Abstract

We define and study the problem of predicting the solution to a linear program (LP) given only partial information about its objective and constraints. This generalizes the problem of learning to predict the purchasing behavior of a rational agent who has an unknown objective function, that has been studied under the name "Learning from Revealed Preferences". We give mistake bound learning algorithms in two settings: in the first, the *objective* of the LP is known to the learner but there is an arbitrary, fixed set of constraints which are unknown. Each example is defined by an additional known constraint and the goal of the learner is to predict the optimal solution of the LP given the union of the known and unknown constraints. This models the problem of predicting the behavior of a rational agent whose *goals* are known, but whose resources are unknown. In the second setting, the objective of the LP is unknown, and changing in a controlled way. The *constraints* of the LP may also change every day, but are known. An example is given by a set of constraints and partial information about the objective, and the task of the learner is again to predict the optimal solution of the partially known LP.

1 Introduction

We initiate the systematic study of a general class of multi-dimensional prediction problems, where the learner wishes to predict the solution to an unknown linear program (LP), given some partial information about either the set of constraints or the objective. In the special case in which there is a single known *constraint* that is changing and the objective that is unknown and fixed, this problem has been studied under the name *learning from revealed preferences* [1, 2, 3, 16] and captures the following scenario: a buyer, with an unknown linear utility function over d goods $u : \mathbb{R}^d \to \mathbb{R}$ defined as $u(\mathbf{x}) = \mathbf{c} \cdot \mathbf{x}$ faces a purchasing decision every day. On day t, she observes a set of prices $\mathbf{p}^t \in \mathbb{R}^d_{\geq 0}$ and buys the bundle of goods that maximizes her unknown utility, subject to a *budget* b:

$$\mathbf{x}^{(t)} = \operatorname*{argmax}_{\mathbf{x}} \mathbf{c} \cdot \mathbf{x} \qquad \text{such that } \mathbf{p}^t \cdot \mathbf{x} \leq b$$

In this problem, the goal of the learner is to predict the bundle that the buyer will buy, given the prices that she faces. Each example at day t is specified by the vector $\mathbf{p}^t \in \mathbb{R}^d_{\geq 0}$ (which fixes the constraint), and the goal is to accurately predict the purchased bundle $\mathbf{x}^{(t)} \in [0, 1]^d$ that is the result of optimizing the unknown linear objective.

It is also natural to consider the class of problems in which the goal is to predict the outcome to a LP broadly e.g. suppose the objective $\mathbf{c} \cdot \mathbf{x}$ is known but there is an *unknown* set of constraints $A\mathbf{x} \leq \mathbf{b}$. An *instance* is again specified by a changing known constraint $(\mathbf{p}^t, b^t)$ and the goal is to predict:

$$\mathbf{x}^{(t)} = \operatorname*{argmax}_{\mathbf{x}} \mathbf{c} \cdot \mathbf{x} \qquad \text{such that } A\mathbf{x} \leq \mathbf{b} \text{ and } \mathbf{p}^t \cdot \mathbf{x} \leq b^t. \tag{1}$$

This models the problem of predicting the behavior of an agent whose *goals* are known, but whose resource constraints are unknown.

Another natural generalization is the problem in which the objective is unknown, and may vary in a specified way across examples, and in which there may also be multiple arbitrary known constraints which vary across examples. Specifically, suppose that there are n distinct, unknown linear objective functions $\mathbf{v}^1, \ldots, \mathbf{v}^n$. An *instance* on day t is specified by a subset of the unknown objective functions, $S^t \subseteq [n] := \{1, \ldots, n\}$ and a convex feasible region $\mathcal{P}^t$, and the goal is to predict:

$$\mathbf{x}^{(t)} = \operatorname*{argmax}_{\mathbf{x}} \sum_{i \in S^t} \mathbf{v}^i \cdot \mathbf{x} \qquad \text{such that } \mathbf{x} \in \mathcal{P}^t. \tag{2}$$

When the changing feasible regions $\mathcal{P}^t$ correspond simply to varying prices as in the revealed preferences problem, this models a setting in which at different times, purchasing decisions are made by different members of an organization, with heterogeneous preferences — but are still bound by an organization-wide budget. The learner's problem is, given the subset of decision makers and the prices at day t, to predict which bundle they will purchase. This generalizes some of the preference learning problems recently studied by Blum et al [6]. Of course, in this generality, we may also consider a richer set of changing constraints which represent things beyond prices and budgets.

In all of the settings we study, the problem can be viewed as the task of predicting the behavior of a rational decision maker, who always chooses the action that maximizes her objective function subject to a set of constraints. Some part of her optimization problem is unknown, and the goal is to learn, through observing her behavior, that unknown part of her optimization problem sufficiently so that we may reliably predict her future actions.

1.1 Our Results

We study both variants of the problem (see below) in the strong *mistake bound* model of learning [13]. In this model, the learner encounters an arbitrary adversarially chosen sequence of examples online and must make a prediction for the optimal solution in each example before seeing future examples. Whenever the learner's prediction is incorrect, the learner encounters a *mistake*, and the goal is to prove an upper bound on the number of mistakes the learner can make, in the worst case over the sequence of examples. Mistake bound learnability is stronger than (and implies) PAC learnability [15].

Known Objective and Unknown Constraints We first study this problem under the assumption that there is a uniform upper bound on the number of bits of precision used to specify the constraint defining each example. In this case, we show that there is a learning algorithm with both running time and mistake bound linear in the number of edges of the polytope formed by the unknown constraint matrix $A\mathbf{x} \leq b$. We note that this is always polynomial in the dimension d when the number of unknown constraints is at most $d + O(1)$. (In Appendix A, we show that by allowing the learner to run in time exponential in d, we can give a mistake bound that is always linear in the dimension and the number of rows of A, but we leave as an open question whether or not this mistake bound can be achieved by an efficient algorithm.) We then show that our bounded precision assumption is necessary — i.e. we show that when the precision to which constraints are specified need not be uniformly upper bounded, then no algorithm for this problem in dimension $d \geq 3$ can have a finite mistake bound.

This lower bound motivates us to study a PAC style variant of the problem, where the examples are not chosen in an adversarial manner, but instead are drawn independently at random from an arbitrary unknown distribution. In this setting, we show that even if the constraints can be specified to arbitrary (even infinite) precision, there is a learner that requires sample complexity only linear in the number of edges of the unknown constraint polytope. This learner can be implemented efficiently when the constraints are specified with finite precision.

Known Constraints and Unknown Objective For the variant of the problem in which the objective is unknown and changing and the constraints are known but changing, we give an algorithm that has a mistake bound and running time polynomial in the dimension d. Our algorithm uses the Ellipsoid algorithm to learn the coefficients of the unknown objective by implementing a separation oracle that generates separating hyperplanes given examples on which our algorithm made a mistake.

We leave the study of either of our problems under natural relaxations (e.g. under a less demanding loss function) and whether it is possible to substantially improve our results in these relaxations as an interesting open problem.

1.2 Related Work

Beigman and Vohra [3] were the first to study *revealed preference problems* (RPP) as a learning problems and to relate them to multi-dimensional classification. They derived sample complexity bounds for such problems by computing the fat shattering dimension of the class of target utility functions, and showed that the set of Lipschitz-continuous valuation functions had finite fat-shattering dimension. Zadimoghaddam and Roth [16] gave efficient algorithms with polynomial sample complexity for PAC learning of the RPP over the class of linear (and piecewise linear) utility functions. Balcan et al. [2] showed a connection between RPP and the structured prediction problem of learning d-dimensional linear classes [7, 8, 12], and use an efficient variant of the compression techniques given by Daniely and Shalev-Shwartz [9] to give efficient PAC algorithms with optimal sample complexity for various classes of economically meaningful utility functions. Amin et al. [1] study the RPP for linear valuation functions in the mistake bound model, and in the query model in which the learner gets to set prices and wishes to maximize profit. Roth et al. [14] also study the query model of learning and give results for strongly concave objective functions, leveraging an algorithm of Belloni et al. [4] for bandit convex optimization with adversarial noise.

All of the works above focus on the setting of predicting the optimizer of a fixed unknown objective function, together with a single known, changing constraint representing prices. This is the primary point of departure for our work — we give algorithms for the more general settings of predicting the optimizer of a LP when there may be many unknown constraints, or when the unknown objective function is changing. Finally, the literature on *preference learning* (see e.g. [10]) has similar goals, but is technically quite distinct: the canonical problem in preference learning is to learn a *ranking* on distinct elements. In contrast, the problem we consider here is to predict the outcome of a continuous optimization problem as a function of varying constraints.

2 Model and Preliminaries

We first formally define the geometric notions used throughout this paper. A *hyperplane* and a *halfspace* in $\mathbb{R}^d$ are the set of points satisfying the linear equation $a_1 x_1 + \ldots a_d x_d = b$ and the linear inequality $a_1 x_1 + \ldots + a_d x_d \leq b$ for a set of a_is respectively, assuming that not all a_i's are simultaneously zero. A set of *hyperplanes* are *linearly independent* if the normal vectors to the hyperplanes are linearly independent. A *polytope* (denoted by $\mathcal{P} \subseteq \mathbb{R}^d$) is the *bounded* intersection of finitely many halfspaces, written as $\mathcal{P} = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$. An *edge-space* e of a polytope $\mathcal{P}$ is a one dimensional subspace that is the intersection of $d - 1$ linearly independent hyperplanes of $\mathcal{P}$, and an *edge* is the intersection between an edge-space e and the polytope $\mathcal{P}$. We denote the set of edges of polytope $\mathcal{P}$ by $E_\mathcal{P}$. A *vertex* of $\mathcal{P}$ is a point where d linearly independent hyperplanes of $\mathcal{P}$ intersect. Equivalently, $\mathcal{P}$ can be written as the *convex hull* of its vertices V denoted by $\mathrm{Conv}(V)$. Finally, we define a set of points to be *collinear* if there exists a line that contains all the points in the set.

We study an online prediction problem with the goal of predicting the optimal solution of a changing LP whose parameters are only partially known. Formally, in each day $t = 1, 2, \ldots$ an adversary chooses a LP specified by a polytope $\mathcal{P}^{(t)}$ (a set of linear inequalities) and coefficients $\mathbf{c}^{(t)} \in \mathbb{R}^d$ of the linear objective function. The learner's goal is to predict the solution $\mathbf{x}^{(t)}$ where $\mathbf{x}^{(t)} = \mathrm{argmax}_{\mathbf{x} \in \mathcal{P}^{(t)}} \mathbf{c}^{(t)} \cdot \mathbf{x}$. After making the prediction $\hat{\mathbf{x}}^{(t)}$, the learner observes the optimal $\mathbf{x}^{(t)}$ and learns whether she has made a mistake ($\hat{\mathbf{x}}^{(t)} \neq \mathbf{x}^{(t)}$). The mistake bound is defined as follows.

Definition 1. Given a LP with feasible polytope $\mathcal{P}$ and objective function $\mathbf{c}$, let $\sigma^{(t)}$ denote the parameters of the LP that are revealed to the learner on day t. A learning algorithm $\mathcal{A}$ takes as input the sequence $\{\sigma^{(t)}\}_t$, the known parameters of an adaptively chosen sequence $\{(\mathcal{P}^{(t)}, \mathbf{c}^{(t)})\}_t$ of LPs and outputs a sequence of predictions $\{\hat{\mathbf{x}}^{(t)}\}_t$. We say that $\mathcal{A}$ has mistake bound M if $\max_{\{(\mathcal{P}^{(t)}, \mathbf{c}^{(t)})\}_t} \left\{ \Sigma_{t=1}^\infty \mathbf{1} \left[\hat{\mathbf{x}}^{(t)} \neq \mathbf{x}^{(t)} \right] \right\} \leq M$, where $\mathbf{x}^{(t)} = \mathrm{argmax}_{\mathbf{x} \in \mathcal{P}^{(t)}} \mathbf{c}^{(t)} \cdot \mathbf{x}$ on day t.

We consider two different instances of the problem described above. First, in Section 3, we study the problem given in (1) in which $\mathbf{c}^{(t)} = \mathbf{c}$ is fixed and known to the learner but the polytope $\mathcal{P}^{(t)} =$

$\mathcal{P} \cap \mathcal{N}^{(t)}$ consists of an unknown fixed polytope $\mathcal{P}$ and a new constraint $\mathcal{N}^{(t)} = \{\mathbf{x} \mid \mathbf{p}^{(t)} \cdot \mathbf{x} \le b^{(t)}\}$ which is revealed to the learner on day t i.e. $\sigma^{(t)} = (\mathcal{N}^{(t)}, \mathbf{c})$. We refer to this as the *Known Objective* problem. Then, in Section 4, we study the problem in which the polytope $\mathcal{P}^{(t)}$ is changing and known but the objective function $\mathbf{c}^{(t)} = \sum_{i \in S^{(t)}} \mathbf{v}^i$ is unknown and changing as in (2) where the set $S^{(t)}$ is known i.e. $\sigma^{(t)} = (\mathcal{P}^{(t)}, S^{(t)})$. We refer to this as the *Known Constraints* problem.

In order for our prediction problem to be well defined, we make Assumption 1 about the observed solution $\mathbf{x}^{(t)}$ in each day. Assumption 1 guarantees that each solution is on a vertex of $\mathcal{P}^{(t)}$.

Assumption 1. *The optimal solution to the LP:* $\max_{\mathbf{x} \in \mathcal{P}^{(t)}} \mathbf{c}^{(t)} \cdot \mathbf{x}$ *is unique for all t.*

3 The Known Objective Problem

In this section, we focus on the *Known Objective Problem* where the coefficients of the objective function $\mathbf{c}$ are fixed and known to the learner but the feasible region $\mathcal{P}^{(t)}$ on day t is unknown and changing. In particular, $\mathcal{P}^{(t)}$ is the intersection of a fixed and unknown polytope $\mathcal{P} = \{\mathbf{x} \mid A\mathbf{x} \le \mathbf{b}, A \subseteq \mathbb{R}^{m \times d}\}$ and a known halfspace $\mathcal{N}^{(t)} = \{\mathbf{x} \mid \mathbf{p}^{(t)} \cdot \mathbf{x} \le b^{(t)}\}$ i.e. $\mathcal{P}^{(t)} = \mathcal{P} \cap \mathcal{N}^{(t)}$.

Throughout this section we make the following assumptions. First, we assume w.l.o.g. (up to scaling) that the points in $\mathcal{P}$ have ℓ_∞-norm bounded by 1.

Assumption 2. *The unknown polytope $\mathcal{P}$ lies inside the unit ℓ_∞-ball i.e. $\mathcal{P} \subseteq \{\mathbf{x} \mid ||\mathbf{x}||_\infty \le 1\}$.*

We also assume that the coordinates of the vertices in $\mathcal{P}$ can be written with finite precision (this is implied if the halfspaces defining $\mathcal{P}$ can be described with finite precision). [1]

Assumption 3. *The coordinates of each vertex of $\mathcal{P}$ can be written with N bits of precision.*

We show in Section 3.3 that Assumption 3 is necessary — without any upper bound on precision, there is no algorithm with a finite mistake bound. Next, we make some non-degeneracy assumptions on polytopes $\mathcal{P}$ and $\mathcal{P}^{(t)}$, respectively. We require these assumptions to hold on each day.

Assumption 4. *Any subset of $d-1$ rows of A have rank $d-1$ where A is the constraint matrix in $\mathcal{P} = \{\mathbf{x} \mid A\mathbf{x} \le \mathbf{b}\}$.*

Assumption 5. *Each vertex of $\mathcal{P}^{(t)}$ is the intersection of exactly d-hyperplanes of $\mathcal{P}^{(t)}$.*

The rest of this section is organized as follows. We present `LearnEdge` for the Known Objective Problem and analyze its mistake bound in Sections 3.1 and 3.2, respectively. Then in Section 3.3, we prove the necessity of Assumption 3 to get a finite mistake bound. Finally in Section 3.4, we present the `LearnHull` in a PAC style setting where the new constraint each day is drawn i.i.d. from an unknown distribution, rather than selected adversarially.

3.1 `LearnEdge` Algorithm

In this section we introduce `LearnEdge` and show in Theorem 1 that the number of mistakes of `LearnEdge` depends linearly on the number of edges $E_\mathcal{P}$ and the precision parameter N and only logarithmically on the dimension d. We defer all the proofs of this section to Appendix B.

Theorem 1. *The number of mistakes and per day running time of `LearnEdge` in the Known Objective Problem are $O(|E_\mathcal{P}|N \log(d))$ and $\mathrm{poly}(m, d, |E_\mathcal{P}|)$ respectively when $A \subseteq \mathbb{R}^{m \times d}$.*

At a high level, `LearnEdge` maintains a set of *prediction information* $\mathcal{I}^{(t)}$ about the prediction history up to day t, and makes prediction in each day based on $\mathcal{I}^{(t)}$ and a set of *prediction rules* (**P.1** − **P.4**). After making a mistake, `LearnEdge` updates the information with a set of *update rules* (**U.1** − **U.4**). The framework of `LearnEdge` is presented in Algorithm 1. We will now present the details of each component.

---

[1] Lemma 6.2.4 from Grotschel et al. [11] states that if each constraint in $\mathcal{P} \subseteq \mathbb{R}^d$ has encoding length at most N then each vertex of $\mathcal{P}$ has encoding length at most $4d^2 N$. Typically the finite precision assumption is made on the constraints of the LP. However, since this assumption implies that the vertices can be described with finite precision, for simplicity, we make our assumption directly on the vertices.

Algorithm 1 Learning in Known Objective Problem (`LearnEdge`)

---

procedure LEARNEDGE($\{\mathcal{N}^{(t)}, \mathbf{x}^{(t)}\}_t$)   ▷ Against adaptive adversary
    Initialize $\mathcal{I}^{(1)}$ to be empty.   ▷ Initialize
    for $t = 1, 2, \cdots$ **do**
        Predict $\hat{\mathbf{x}}^{(t)}$ according to one of **P.1** - **P.4** based on $\mathcal{N}^{(t)}$ and $\mathcal{I}^{(t)}$.   ▷ Predict
        if $\hat{\mathbf{x}}^{(t)} \neq \mathbf{x}^{(t)}$ **then**
            $X^{(t+1)} \leftarrow X^{(t)} \cup \{\mathbf{x}^{(t)}\}$.
            Update $\mathcal{I}^{(t+1)}$ with **U.1** - **U.4** based on $\mathbf{x}^{(t)}$.   ▷ Update
end procedure

Prediction Information   It is natural to ask "What information is useful for prediction?" Lemma 2 establishes the importance of the set of edges $E_{\mathcal{P}}$ by showing that all the observed solutions will be on an element of $E_{\mathcal{P}}$.

Lemma 2. *On any day t, the observed solution* $\mathbf{x}^{(t)}$ *lies on an edge in* $E_{\mathcal{P}}$.

In the proof of Lemma 2 we also show that when $\mathbf{x}^{(t)}$ does not bind the new constraint $\mathcal{N}^{(t)}$, then $\mathbf{x}^{(t)}$ is the solution for the underlying LP: $\mathrm{argmax}_{\mathbf{x}\in\mathcal{P}} \mathbf{c} \cdot \mathbf{x}$.

Corollary 1. *If* $\mathbf{x}^{(t)} \in \{\mathbf{x} \mid \mathbf{p}^{(t)}\mathbf{x} < b^{(t)}\}$ *then* $\mathbf{x}^{(t)} = \mathbf{x}^* \equiv \mathrm{argmax}_{\mathbf{x}\in\mathcal{P}} \mathbf{c} \cdot \mathbf{x}$.

We then show how an edge-space e of $\mathcal{P}$ can be recovered after seeing 3 collinear observed solutions.

Lemma 3. *Let* $\mathbf{x}, \mathbf{y}, \mathbf{z}$ *be 3 distinct collinear points on edges of* $\mathcal{P}$. *Then they are all on the* same *edge of* $\mathcal{P}$ *and the 1-dimensional subspace containing them is an edge-space of* $\mathcal{P}$.

Given the relation between observed solutions and edges, the information $\mathcal{I}^{(t)}$ is stored as follows:
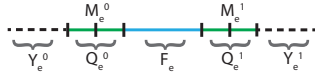

Figure 1: Regions on an edge-space e: feasible region F_e (blue), questionable intervals Q_e^0 and Q_e^1 (green) with their mid-points M_e^0 and M_e^1 and infeasible regions Y_e^0 and Y_e^1 (dashed).

I.1 (**Observed Solutions**) `LearnEdge` keeps track of the set of observed solutions that were predicted incorrectly so far $X^{(t)} = \{\mathbf{x}^{(\tau)} : \tau \leq t \quad \hat{\mathbf{x}}^{(\tau)} \neq \mathbf{x}^{(\tau)}\}$ and also the solution for the underlying unknown polytope $\mathbf{x}^* \equiv \mathrm{argmax}_{\mathbf{x}\in\mathcal{P}} \mathbf{c} \cdot \mathbf{x}$ if it is observed.

I.2 (**Edges**) `LearnEdge` keeps track of the set of edge-spaces $E^{(t)}$ given by any 3 collinear points in $X^{(t)}$. For each $e \in E^{(t)}$, it also maintains the regions on e that are certainly *feasible* or *infeasible*. The remaining parts of e called the *questionable* region is where `LearnEdge` cannot classify as infeasible or feasible with certainty (see Figure 1). Formally,

1. (**Feasible Interval**) The *feasible interval* F_e is an interval along e that is identified to be on the boundary of $\mathcal{P}$. More formally, $F_e = \mathrm{Conv}(X^{(t)} \cap e)$.

2. (**Infeasible Region**) The *infeasible region* $Y_e = Y_e^0 \cup Y_e^1$ is the union of two disjoint intervals Y_e^0 and Y_e^1 that are identified to be outside of $\mathcal{P}$. By Assumption 2, we initialize the infeasible region Y_e to $\{x \in e \mid \|x\|_\infty > 1\}$ for all e.

3. (**Questionable Region**) The *questionable region* $Q_e = Q_e^0 \cup Q_e^1$ on e is the union of two disjoint *questionable intervals* along e. Formally, $Q_e = e \setminus (F_e \cup Y_e)$. The points in Q_e cannot be certified to be either inside or outside of $\mathcal{P}$ by `LearnEdge`.

4. (**Midpoints in** Q_e) For each questionable interval Q_e^i, let M_e^i denote the midpoint of Q_e^i.

We add the superscript (t) to show the dependence of these quantities on days. Furthermore, we eliminate the subscript e when taking the union over all elements in $E^{(t)}$, e.g. $F^{(t)} = \bigcup_{e\in E^{(t)}} F_e^{(t)}$. So the information $\mathcal{I}^{(t)}$ can be written as follows: $\mathcal{I}^{(t)} = \left(X^{(t)}, E^{(t)}, F^{(t)}, Y^{(t)}, Q^{(t)}, M^{(t)}\right)$.

Prediction Rules   We now focus on the prediction rules of `LearnEdge`. On day t, let $\widetilde{\mathcal{N}}^{(t)} = \{\mathbf{x} \mid \mathbf{p}^{(t)} \cdot \mathbf{x} = b^{(t)}\}$ be the hyperplane specified by the additional constraint $\mathcal{N}^{(t)}$. If $\mathbf{x}^{(t)} \notin \widetilde{\mathcal{N}}^{(t)}$, then $\mathbf{x}^{(t)} = \mathbf{x}^*$ by Corollary 1. So whenever the algorithm observes $\mathbf{x}^*$, it will store $\mathbf{x}^*$ and predict it in the future days when $\mathbf{x}^* \in \mathcal{N}^{(t)}$. This is case **P.1**. So in the remaining cases we know $\mathbf{x}^* \notin \mathcal{N}^{(t)}$.

The analysis of Lemma 2 shows that $\mathbf{x}^{(t)}$ must be in the intersection between $\widetilde{\mathcal{N}}^{(t)}$ and the edges $E_{\mathcal{P}}$, so $\mathbf{x}^{(t)} = \mathrm{argmax}_{\mathbf{x} \in \widetilde{\mathcal{N}}^{(t)} \cap E_{\mathcal{P}}} \mathbf{c} \cdot \mathbf{x}$. Hence, `LearnEdge` can restrict its prediction to the following *candidate* set: $\mathrm{Cand}^{(t)} = \{(E^{(t)} \cup X^{(t)}) \setminus \bar{E}^{(t)}\} \cap \widetilde{\mathcal{N}}^{(t)}$ where $\bar{E}^{(t)} = \{e \in E^{(t)} \mid e \subseteq \widetilde{\mathcal{N}}^{(t)}\}$. As we show in Lemma 4, $\mathbf{x}^{(t)}$ will not be in $\bar{E}^{(t)}$, so it is safe to remove $\bar{E}^{(t)}$ from $\mathrm{Cand}^{(t)}$.

Lemma 4. *Let e be an edge-space of $\mathcal{P}$ such that $e \subseteq \widetilde{\mathcal{N}}^{(t)}$, then $\mathbf{x}^{(t)} \notin e$.*

However, $\mathrm{Cand}^{(t)}$ can be empty or only contain points in the infeasible regions of the edge-spaces. If so, then there is simply not enough information to predict a feasible point in $\mathcal{P}$. Hence, `LearnEdge` predicts an arbitrary point outside of $\mathrm{Cand}^{(t)}$. This is case **P.2**.

Otherwise $\mathrm{Cand}^{(t)}$ contains points from the feasible and questionable regions of the edge-spaces. `LearnEdge` predicts from a subset of $\mathrm{Cand}^{(t)}$ called the *extended feasible region* $\mathrm{Ext}^{(t)}$ instead of directly predicting from $\mathrm{Cand}^{(t)}$. $\mathrm{Ext}^{(t)}$ contains the whole feasible region and only parts of the questionable region on all the edge-spaces in $E^{(t)} \setminus \bar{E}^{(t)}$. We will show later that this guarantees `LearnEdge` makes progress in learning the true feasible region on some edge-space upon making a mistake. More formally, $\mathrm{Ext}^{(t)}$ is the intersection of $\widetilde{\mathcal{N}}^{(t)}$ with the union of intervals between the two mid-points $(M_e^0)^{(t)}$ and $(M_e^1)^{(t)}$ on every edge-space $e \in E^{(t)} \setminus \bar{E}^{(t)}$ and all points in $X^{(t)}$: $\mathrm{Ext}^{(t)} = \{X^{(t)} \cup \{\cup_{e \in E^{(t)} \setminus \bar{E}^{(t)}} \mathrm{Conv}\left((M_e^0)^{(t)}, (M_e^1)^{(t)}\right)\}\} \cap \widetilde{\mathcal{N}}^{(t)}$.

In **P.3**, if $\mathrm{Ext}^{(t)} \neq \emptyset$ then `LearnEdge` predicts the point with the highest objective value in $\mathrm{Ext}^{(t)}$.

Finally, if $\mathrm{Ext}^{(t)} = \emptyset$, then we know $\widetilde{\mathcal{N}}^{(t)}$ only intersects within the questionable regions of the learned edge-spaces. In this case, `LearnEdge` predicts the intersection point with the lowest objective value, which corresponds to **P.4**. Although it might seem counter-intuitive to predict the point with the lowest objective value, this guarantees that `LearnEdge` makes progress in learning the true feasible region on some edge-space upon making a mistake. The prediction rules are summarized as follows:

P.1 First, if $\mathbf{x}^*$ is observed and $\mathbf{x}^* \in \mathcal{N}^{(t)}$, then predict $\hat{\mathbf{x}}^{(t)} \leftarrow \mathbf{x}^*$;

P.2 Else if $\mathrm{Cand} = \emptyset$ or $\mathrm{Cand}^{(t)} \subseteq \bigcup_{e \in E^{(t)}} Y_e^{(t)}$, then predict any point outside $\mathrm{Cand}^{(t)}$;

P.3 Else if $\mathrm{Ext}^{(t)} \neq \emptyset$, then predict $\hat{\mathbf{x}}^{(t)} = \mathrm{argmax}_{\mathbf{x} \in \mathrm{Ext}^{(t)}} \mathbf{c} \cdot \mathbf{x}$;

P.4 Else, predict $\hat{\mathbf{x}}^{(t)} = \mathrm{argmin}_{\mathbf{x} \in \mathrm{Cand}^{(t)}} \mathbf{c} \cdot \mathbf{x}$.

Update Rules   Next we describe how `LearnEdge` updates its information. Upon making a mistake, `LearnEdge` adds $\mathbf{x}^{(t)}$ to the set of previously observed solutions $X^{(t)}$ i.e. $X^{(t+1)} \leftarrow X^{(t)} \cup \{\mathbf{x}^{(t)}\}$. Then it performs one of the following four mutually exclusive update rules (**U.1**-**U.4**) in order.

U.1 If $\mathbf{x}^{(t)} \notin \widetilde{\mathcal{N}}^{(t)}$, then `LearnEdge` records $\mathbf{x}^{(t)}$ as the unconstrained optimal solution $\mathbf{x}^*$.

U.2 Then if $\mathbf{x}^{(t)}$ is not on any learned edge-space in $E^{(t)}$, `LearnEdge` will try to learn a new edge-space by checking the collinearity of $\mathbf{x}^{(t)}$ and any couple of points in $X^{(t)}$. So after this update `LearnEdge` might recover a new edge-space of the polytope.

If the previous updates were not invoked, then $\mathbf{x}^{(t)}$ was on some learned edge-space e. `LearnEdge` then compares the objective values of $\hat{\mathbf{x}}^{(t)}$ and $\mathbf{x}^{(t)}$ (we know $\mathbf{c} \cdot \hat{\mathbf{x}}^{(t)} \neq \mathbf{c} \cdot \mathbf{x}^{(t)}$ by Assumption 1):

U.3 If $\mathbf{c} \cdot \hat{\mathbf{x}}^{(t)} > \mathbf{c} \cdot \mathbf{x}^{(t)}$, then $\hat{\mathbf{x}}^{(t)}$ must be infeasible and `LearnEdge` then updates the questionable and infeasible regions for e.

U.4 If $\mathbf{c} \cdot \hat{\mathbf{x}}^{(t)} < \mathbf{c} \cdot \mathbf{x}^{(t)}$ then $\mathbf{x}^{(t)}$ was outside of the extended feasible region of e. `LearnEdge` then updates the questionable region and feasible interval on e.

In both of **U.3** and **U.4**, `LearnEdge` will shrink some questionable interval substantially till the interval has length less than 2^{-N} in which case Assumption 3 implies that the interval contains no points. So `LearnEdge` can update the adjacent feasible region and infeasible interval accordingly.

3.2 Analysis of `LearnEdge`

Whenever `LearnEdge` makes a mistake, one of the update rules **U.1** - **U.4** is invoked. So the number of mistakes of `LearnEdge` is bounded by the number of times each update rule is invoked. The mistake bound of `LearnEdge` in Theorem 1 is hence the sum of mistakes bounds in Lemmas 5-7.

Lemma 5. *Update **U.1** is invoked at most 1 time.*

Lemma 6. *Update **U.2** is invoked at most $3|E_{\mathcal{P}}|$ times.* [2]

Lemma 7. *Updates **U.3** and **U.4** are invoked at most $O(|E_{\mathcal{P}}|N \log(d))$ times.*

3.3 Necessity of the Precision Bound

We show the necessity of Assumption 3 by showing that the dependence on the precision parameter N in our mistake bound is tight. We show that subject to Assumption 3, there exist a polytope and a sequence of additional constraints such that any learning algorithm will make $\Omega(N)$ mistakes. This implies that without any upper bound on precision, it is impossible to learn with finite mistakes.

Theorem 8. *For any learning algorithm $\mathcal{A}$ in the Known Objective Problem and any $d \geq 3$, there exists a polytope $\mathcal{P}$ and a sequence of additional constraints $\{\mathcal{N}^{(t)}\}_t$ such that the number of mistakes made by $\mathcal{A}$ is at least $\Omega(N)$.* [3]

3.4 Stochastic Setting

Given the lower bound in Theorem 8, we ask "In what settings we can still learn without an upper bound on the precision to which constraints are specified?" The lower bound implies we must abandon the adversarial setting so we consider a PAC style variant. In this variant, the additional constraint at each day t is drawn i.i.d. from some fixed but unknown distribution $\mathcal{D}$ over $\mathbb{R}^d \times \mathbb{R}$ such that each point $(\mathbf{p}, b)$ drawn from $\mathcal{D}$ corresponds to the halfspace $\mathcal{N} = \{\mathbf{x} \mid \mathbf{p} \cdot \mathbf{x} \leq b\}$. We make no assumption on the form of $\mathcal{D}$ and require our bounds to hold in the worst case over all choices of $\mathcal{D}$.

We describe `LearnHull` an algorithm based on the following high level idea: `LearnHull` keeps track of the convex hull $\mathcal{C}^{(t-1)}$ of all the solutions observed up to day t. `LearnHull` then behaves as if this convex hull is the entire feasible region. So at day t, given the constraint $\mathcal{N}^{(t)} = \{\mathbf{x} \mid \mathbf{p}^{(t)} \cdot \mathbf{x} \leq b^{(t)}\}$, `LearnHull` predicts $\hat{\mathbf{x}}^{(t)}$ where

$$\hat{\mathbf{x}}^{(t)} = \text{argmax}_{\mathbf{x} \in \mathcal{C}^{(t-1)} \cap \mathcal{N}^{(t)}} \ \mathbf{c} \cdot \mathbf{x}. \tag{3}$$

`LearnHull`'s hypothetical feasible region is therefore always a subset of the true feasible region – i.e. it can never make a mistake because its prediction was infeasible, but only because its prediction was sub-optimal. Hence, whenever `LearnHull` makes a mistake, it must have observed a point that expands the convex hull. Hence, whenever it fails to predict $\mathbf{x}^{(t)}$, `LearnHull` will enlarge its feasible region by adding the point $\mathbf{x}^{(t)}$ to the convex hull:

$$\mathcal{C}^{(t)} \leftarrow \text{Conv}(\mathcal{C}^{(t-1)} \cup \{\mathbf{x}^{(t)}\}), \tag{4}$$

otherwise it will simply set $\mathcal{C}^{(t)} \leftarrow \mathcal{C}^{(t-1)}$ for the next day. `LearnHull` is described formally in Algorithm 2.

We show that the expected number of mistakes of `LearnHull` over T days is linear in the number of edges of $\mathcal{P}$ and only logarithmic in T. [4]

---

[2] The dependency on $|E_{\mathcal{P}}|$ can be improved by replacing it with the set of edges of $\mathcal{P}$ on which an optimal solution is observed. This applies to all the dependencies on $|E_{\mathcal{P}}|$ in our bounds.

[3] We point out that the condition $d \geq 3$ is necessary in the statement of Theorem 8 since there exists learning algorithms for $d = 1$ and $d = 2$ with finite mistake bounds independent of N. See Appendix C.

[4] `LearnHull` can be implemented efficiently in time $\text{poly}(T, N, d)$ if all of the coefficients in the unknown constraints in $\mathcal{P}$ are represented in N bits. Note that given the observed solutions so far and a new point, a separation oracle can be implemented in time $\text{poly}(T, N, d)$ using a LP solver.

Algorithm 2 Stochastic Procedure (`LearnHull`)
---
    procedure LEARNHULL($\mathcal{D}$)
        $\mathcal{C}^{(0)} \leftarrow \emptyset$. $\triangleright$ Initialize
        Observe $\mathcal{N}^{(t)} \sim D$ and set $\hat{\mathbf{x}}^{(t)}$ as in (3). $\triangleright$ Predict
        Observers $\mathbf{x}^{(t)} = \mathrm{argmax}_{\mathbf{x} \in \mathcal{P} \cap \mathcal{N}^{(t)}} \, \mathbf{c} \cdot \mathbf{x}$ and update $\mathcal{C}^{(t)}$ as in (4). $\triangleright$ Update
    end procedure

Theorem 9. *For any $T > 0$ and any constraint distribution $\mathcal{D}$, the expected number of mistakes of* `LearnHull` *after T days is bounded by $O\left(|E_{\mathcal{P}}| \log(T)\right)$.*

To prove Theorem 9, first in Lemma 10 we bound the probability that the solution observed at day t falls outside of the convex hull of the previously observed solutions. This is the only event that can cause `LearnHull` to make a mistake. In Lemma 10, we abstract away the fact that the point observed at each day is the solution to some optimization problem.

Lemma 10. *Let $\mathcal{P}$ be a polytope and $\mathcal{D}$ a distribution over points on $E_{\mathcal{P}}$. Let $X = \{x_1, \ldots, x_{t-1}\}$ be $t-1$ i.i.d. draws from $\mathcal{D}$ and x_t an additional independent draw from $\mathcal{D}$. Then $\Pr[x_t \notin \mathrm{Conv}(X)] \leq 2|E_{\mathcal{P}}|/t$ where the probability is taken over the draws of points $x_1, \ldots, x_t$ from $\mathcal{D}$.*

Finally in Theorem 11 we convert the bound on the expected number of mistakes of `LearnHull` in Theorem 9 to a high probability bound. [5]

Theorem 11. *There exists a deterministic procedure such that after $T = O\left(|E_{\mathcal{P}}| \log\left(1/\delta\right)\right)$ days, the probability (over the randomness of the additional constraint) that the procedure makes a mistake on day $T + 1$ is at most δ for any $\delta \in (0, 1/2)$.*

4 The Known Constraints Problem

We now consider the *Known Constraints Problem* in which the learner observes the changing constraint polytope $\mathcal{P}^{(t)}$ at each day, but does not know the changing objective function which we assume to be written as $\mathbf{c}^{(t)} = \sum_{i \in S^{(t)}} \mathbf{v}^i$, where $\{\mathbf{v}^i\}_{i \in [n]}$ are fixed but unknown. Given $\mathcal{P}^{(t)}$ and the subset $S^{(t)} \subseteq [n]$, the learner must make a prediction $\hat{\mathbf{x}}^{(t)}$ on each day. Inspired by Bhaskar et al. [5], we use the Ellipsoid algorithm to learn the coefficients $\{\mathbf{v}^i\}_{i \in [n]}$, and show that the mistake bound of the resulting algorithm is bounded by the (polynomial) running time of the Ellipsoid. We use $V \in \mathbb{R}^{d \times n}$ to denote the matrix whose columns are $\mathbf{v}^i$ and make the following assumption on V.

Assumption 6. *Each entry in V can be written with N bits of precision. Also w.l.o.g. $||V||_F \leq 1$.*

Similar to Section 3 we assume the coordinates of $\mathcal{P}^{(t)}$'s vertices can be written with finite precision.[6]

Assumption 7. *The coordinates of each vertex of $\mathcal{P}^{(t)}$ can be written with N bits of precision.*

We first observe that the coefficients of the objective function represent a point that is guaranteed to lie in a region $\mathcal{F}$ (described below) which may be written as the intersection of possibly infinitely many halfspaces. Given a subset $S \subseteq [n]$ and a polytope $\mathcal{P}$, let $\mathbf{x}^{S,\mathcal{P}}$ denote the optimal solution to the instance defined by S and $\mathcal{P}$. Informally, the halfspaces defining $\mathcal{F}$ ensure that for any problem instance defined by arbitrary choices of S and $\mathcal{P}$, the objective value of the *optimal* solution $x^{S,\mathcal{P}}$ must be at least as high as the objective value of any *feasible* point in $\mathcal{P}$. Since the convergence rate of the Ellipsoid algorithm depends on the precision to which constraints are specified, we do not in fact consider a hyperplane for every feasible solution but only for those solutions that are vertices of the feasible polytope $\mathcal{P}$. This is not a relaxation, since LPs always have vertex-optimal solutions.

We denote the set of all vertices of polytope $\mathcal{P}$ by $\mathrm{vert}(\mathcal{P})$, and the set of polytopes $\mathcal{P}$ satisfying Assumption 7 by Φ. We then define $\mathcal{F}$ as follows:

$$\mathcal{F} = \left\{ W = (\mathbf{w}^1, \ldots, \mathbf{w}^n) \in \mathbb{R}^{n \times d} \mid \forall S \subseteq [n], \forall \mathcal{P} \in \Phi, \sum_{i \in S} \mathbf{w}^i \cdot \left(\mathbf{x}^{S,\mathcal{P}} - \mathbf{x} \right) \geq 0, \forall \mathbf{x} \in \mathrm{vert}(\mathcal{P}) \right\}$$
(5)

---

[5]`LearnEdge` fails to give any non-trivial mistake bound in the adversarial setting.

[6]We again point out that this is implied if the halfspaces defining the polytope are described with finite precision [11].

The idea behind our `LearnEllipsoid` algorithm is that we will run a copy of the Ellipsoid algorithm with variables $\mathbf{w} \in \mathbb{R}^{d \times n}$, as if we were solving the feasibility LP defined by the constraints defining $\mathcal{F}$. We will always predict according to the centroid of the ellipsoid maintained by the Ellipsoid algorithm (i.e. its candidate solution). Whenever a mistake occurs, we are able to find one of the constraints that define $\mathcal{F}$ such that our prediction violates the constraint – exactly what is needed to take a step in solving the feasibility LP. Since we know $\mathcal{F}$ is non-empty (at least the true objective function V lies within it) we know that the LP we are solving is feasible. Given the polynomial convergence time of the Ellipsoid algorithm, this gives a polynomial mistake bound for our algorithm.

The Ellipsoid algorithm will generate a sequence of ellipsoids with decreasing volume such that each one contains feasible region $\mathcal{F}$. Given the ellipsoid $\mathcal{E}^{(t)}$ at day t, `LearnEllipsoid` uses the centroid of $\mathcal{E}^{(t)}$ as its hypothesis for the objective function $W^{(t)} = ((\mathbf{w}^1)^{(t)}, \dots, (\mathbf{w}^n)^{(t)})$. Given the subset $S^{(t)}$ and polytope $\mathcal{P}^{(t)}$, `LearnEllipsoid` predicts

$$\hat{\mathbf{x}}^{(t)} \in \operatorname*{argmax}_{\mathbf{x} \in \mathcal{P}^{(t)}} \{ \sum_{i \in S^{(t)}} (\mathbf{w}^i)^{(t)} \cdot \mathbf{x} \}. \tag{6}$$

When a mistake occurs, `LearnEllipsoid` finds the hyperplane

$$\mathcal{H}^{(t)} = \left\{ W = (\mathbf{w}^1, \dots, \mathbf{w}^n) \in \mathbb{R}^{n \times d} : \sum_{i \in S^{(t)}} \mathbf{w}^i \cdot (\mathbf{x}^{(t)} - \hat{\mathbf{x}}^{(t)}) > 0 \right\} \tag{7}$$

that separates the centroid of the current ellipsoid (the current candidate objective) from $\mathcal{F}$.

After the update, we use the Ellipsoid algorithm to compute the minimum-volume ellipsoid $\mathcal{E}^{(t+1)}$ that contains $\mathcal{H}^{(t)} \cap \mathcal{E}^{(t)}$. On day $t+1$, `LearnEllipsoid` sets $W^{(t+1)}$ to be the centroid of $\mathcal{E}^{(t+1)}$. The above procedure is formalized in Algorithm 3.

Algorithm 3 Learning with Known Constraints (`LearnEllipsoid`)

---

procedure LEARNELLIPSOID($\mathcal{A}$)   ▷ Against adversary $\mathcal{A}$

$\quad \mathcal{I}^{(1)} \leftarrow \left(\mathcal{E}^{(1)} = \{ \mathbf{z} \in \mathbb{R}^{n \times d} : ||\mathbf{z}||_F \leq 2 \}, W^{(1)} = \text{CENTROID}(\mathcal{E}^{(1)}) \right)$ ▷ Initialize
$\quad$ **for** $t = 1 \dots$ **do**
$\quad\quad$ Given $S^{(t)}, \mathcal{P}^{(t)}$, set $\hat{\mathbf{x}}^{(t)}$ as in (6).   ▷ Predict
$\quad\quad$ **if** $\mathbf{x}^{(t)} \neq \hat{\mathbf{x}}^{(t)}$ **then**
$\quad\quad\quad$ set $\mathcal{H}^{(t)}$ as in (7).   ▷ Update
$\quad\quad\quad \mathcal{E}^{(t+1)} \leftarrow \text{ELLIPSOID}(\mathcal{H}^{(t)}, \mathcal{E}^{(t)}), W^{(t+1)} = \text{CENTROID}(\mathcal{E}^{(t+1)})$.
$\quad\quad\quad \mathcal{I}^{(t+1)} = \left(\mathcal{E}^{(t+1)}, W^{(t+1)} \right)$.
$\quad\quad$ **else**
$\quad\quad\quad \mathcal{I}^{(t+1)} \leftarrow \mathcal{I}^{(t)}$.
end procedure

We left the procedure used to solve the LP in the prediction rule of `LearnEllipsoid` unspecified. To simplify our analysis, we use a specific LP solver to obtain a prediction $\hat{\mathbf{x}}^{(t)}$ which is a vertex of $\mathcal{P}^{(t)}$. We defer all the proofs of this section to Appendix D.

Theorem 12 (Theorem 6.4.12 and Remark 6.5.2 [11]). *There exists a LP solver that runs in time polynomial in the length of its input and returns an exact solution that is a vertex of $\mathcal{P}^{(t)}$.*

In Theorem 13, we show that the number of mistakes made by `LearnEllipsoid` is at most the number of updates that the Ellipsoid algorithm makes before it finds a point in $\mathcal{F}$ and the number of updates of the Ellipsoid algorithm can be bounded by well-known results from the literature on LP.

Theorem 13. *The total number of mistakes and the running time of `LearnEllipsoid` in the Known Constraints Problem is at most $\operatorname{poly}(n, d, N)$.*

References

[1] AMIN, K., CUMMINGS, R., DWORKIN, L., KEARNS, M., AND ROTH, A. Online learning and profit maximization from revealed preferences. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence* (2015), pp. 770–776.

[2] BALCAN, M., DANIELY, A., MEHTA, R., URNER, R., AND VAZIRANI, V. Learning economic parameters from revealed preferences. In *Proceeding of the 10th International Conference on Web and Internet Economics* (2014), pp. 338–353.

[3] BEIGMAN, E., AND VOHRA, R. Learning from revealed preference. In *Proceedings of the 7th ACM Conference on Electronic Commerce* (2006), pp. 36–42.

[4] BELLONI, A., LIANG, T., NARAYANAN, H., AND RAKHLIN, A. Escaping the local minima via simulated annealing: Optimization of approximately convex functions. In *Proceeding of the 28th Conference on Learning Theory* (2015), pp. 240–265.

[5] BHASKAR, U., LIGETT, K., SCHULMAN, L., AND SWAMY, C. Achieving target equilibria in network routing games without knowing the latency functions. In *Proceeding of the 55th IEEE Annual Symposium on Foundations of Computer Science* (2014), pp. 31–40.

[6] BLUM, A., MANSOUR, Y., AND MORGENSTERN, J. Learning what's going on: Reconstructing preferences and priorities from opaque transactions. In *Proceedings of the 16th ACM Conference on Economics and Computation* (2015), pp. 601–618.

[7] COLLINS, M. Discriminative reranking for natural language parsing. In *Proceedings of the 17th International Conference on Machine Learning* (2000), Morgan Kaufmann, pp. 175–182.

[8] COLLINS, M. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing* (2002), pp. 1–8.

[9] DANIELY, A., AND SHALEV-SHWARTZ, S. Optimal learners for multiclass problems. In *Proceedings of the 27th Conference on Learning Theory* (2014), pp. 287–316.

[10] FÜRNKRANZ, J., AND HÜLLERMEIER, E. *Preference learning*. Springer, 2010.

[11] GRÖTSCHEL, M., LOVÁSZ, L., AND SCHRIJVER, A. *Geometric Algorithms and Combinatorial Optimization*, second corrected ed., vol. 2 of *Algorithms and Combinatorics*. Springer, 1993.

[12] LAFFERTY, J., MCCALLUM, A., AND PEREIRA, F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning* (2001), pp. 282–289.

[13] LITTLESTONE, N. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning 2*, 4 (1988), 285–318.

[14] ROTH, A., ULLMAN, J., AND WU, Z. Watch and learn: Optimizing from revealed preferences feedback. In *Proceedings of the 48th Annual ACMSymposium on Theory of Computing* (2016), pp. 949–962.

[15] VALIANT, L. A theory of the learnable. *Communications of the ACM 27*, 11 (1984), 1134–1142.

[16] ZADIMOGHADDAM, M., AND ROTH, A. Efficiently learning from revealed preference. In *Proceedings of the 8th International Workshop on Internet and Network Economics* (2012), pp. 114–127.

A   Polynomial Mistake Bound with Exponential Running Time

In this section we give a simple randomized algorithm for the unknown constraints problem, that in expectation makes a number of mistakes that is only linear in the dimension d, the number of rows in the unknown constraint matrix A (denoted by m), and the bit precision N, but which requires exponential running time. When the number of rows is large, this can represent an exponential improvement over the mistake bound of LearnEdge, which is linear in the number of *edges* on the polytope $\mathcal{P}$ defined by A. This algorithm which we describe shortly is a randomized variant of the well known halving algorithm [13]. We leave it as an open problem whether the mistake bound achieved by this algorithm can also be achieved by a computationally efficient algorithm.

Let $\mathcal{K}$ be the hypothesis class of all polytopes formed by m constraints in d dimensions, such that each entry of each constraint can be written as a multiple of $1/2^N$ (and without loss of generality, up to scaling, has absolute value at most 1). We then have

$$|\mathcal{K}| = 2^{O(dmN)}.$$

We write $\mathcal{K}^{(t)}$ to denote the polytopes that are consistent with the examples and solutions we have seen up to and including day t. Note that $|\mathcal{K}^{(t)}| \geq 1$ for every t because there is some polytope (specifically the true unknown polytope $\mathcal{P}$) that is consistent with all the optimal solutions. On each day t we keep track of consistent polytopes and more specifically update the set of consistent polytopes by

$$\mathcal{K}^{(t+1)} = \left\{ \mathcal{P} \in \mathcal{K}^{(t)} \mid \mathbf{x}^{(t)} \in \underset{\mathbf{x} \in \mathcal{P} \cap \mathcal{N}^{(t)}}{\operatorname{argmax}} \mathbf{c} \cdot \mathbf{x} \right\}, \tag{8}$$

where $\mathcal{N}^{(t)}$ is the new constraint on day t. The formal description of the algorithm, FCP, is presented in Algorithm 4. To predict at each day, FCP selects a polytope $\hat{\mathcal{P}}^{(t)}$ from $\mathcal{K}^{(t)}$ uniformly at random and guesses $\hat{\mathbf{x}}^{(t)}$ that solves the following LP: $\max_{\mathbf{x} \in \hat{\mathcal{P}}^{(t)} \cap \mathcal{N}^{(t)}} \mathbf{c} \cdot \mathbf{x}$.

Algorithm 4 Find Consistent Polytope FCP

procedure FCP
    $\mathcal{K}^{(1)} = \mathcal{K}$. $\triangleright$ Initialize
    for $t = 1 \ldots$ **do**
        Choose $\hat{\mathcal{P}}^{(t)} \in \mathcal{K}^{(t)}$ uniformly at random.
        Guess $\hat{\mathbf{x}}^{(t)} \in \operatorname{argmax}_{\mathbf{x} \in \hat{\mathcal{P}}^{(t)} \cap \mathcal{N}^{(t)}} \mathbf{c} \cdot \mathbf{x}$. $\triangleright$ Predict
        Observe $\mathbf{x}^{(t)}$ and set $\mathcal{K}^{(t+1)}$ as in (8).
end procedure

We now bound the expected number of mistakes that FCP makes.

Theorem 14. *The expected number of mistakes that* FCP *makes is at most* $\log(|\mathcal{K}|) = O(dmN)$, *where the expectation is over the randomness of* FCP *and possible randomness of the adversary.*

Proof. First note that the probability that FCP *does not* make a mistake at day t can be expressed as $|\mathcal{K}^{(t+1)}|/|\mathcal{K}^{(t)}|$. This is because if FCP makes a mistake at day t, it must have selected a polytope that will be eliminated at the next day (also note that FCP selects its polytope from among the consistent set uniformly at random). Now consider the product of these probabilities over all days $t = 1 \ldots T$.

$$\prod_{t=1}^{T} (1 - \mathbb{P}\left[\text{Mistake at day } t\right]) = \prod_{t=1}^{T} \frac{|\mathcal{K}^{(t+1)}|}{|\mathcal{K}^{(t)}|} = \frac{|\mathcal{K}^{(T+1)}|}{|\mathcal{K}^{(1)}|}.$$

Finally, note that the expected number of mistakes is the sum of probabilities of making mistakes over all days. Using the inequality $(1 - x) \leq e^{-x}$ for every $x \in [0, 1]$ and rearranging terms we get

$$\sum_{t=1}^{T} \mathbb{P}\left[\text{Mistake at day } t\right] \leq \log \left(\frac{|\mathcal{K}^{(1)}|}{|\mathcal{K}^{(T+1)}|} \right) \leq O(dmN),$$

since $|\mathcal{K}^{(1)}| = 2^{O(dmN)}$ and $|\mathcal{K}^{(T+1)}| \geq 1$. $\qquad \square$

Finally, we remark that the randomized halving technique above will also result in a polynomial mistake bound in the more demanding variant where not only the underlying constraint matrix but also the linear objective function is unknown. This is because the coefficients of the objective function can be written in dN bits if they are also represented with finite precision. However, the issue about the exponential running time still exists in the new setting.

B Missing Proofs from Section 3

B.1 Section 3.1

Proof of Lemma 2. Let $\mathbf{x}^*$ be the optimal solution of the linear program solved over the unknown polytope $\mathcal{P}$, without the added constraint i.e. $\mathbf{x}^* \equiv \operatorname{argmax}_{x \in \mathcal{P}} \mathbf{c} \cdot \mathbf{x}$.

1. Suppose that $\mathbf{x}^* \in \mathcal{N}^{(t)}$, then clearly $\mathbf{x}^{(t)} = \mathbf{x}^*$. By Assumption 1, $\mathbf{x}^*$ lies on a vertex of $\mathcal{P}$ and therefore $\mathbf{x}^{(t)}$ lies on one of the edges of $\mathcal{P}$.

2. Suppose that $\mathbf{x}^* \notin \mathcal{N}^{(t)}$ i.e. $\mathbf{p}^{(t)} \cdot \mathbf{x}^* > b^{(t)}$. Then we claim that the optimal solution $\mathbf{x}^{(t)}$ satisfies $\mathbf{p}^{(t)} \cdot \mathbf{x}^{(t)} = b^{(t)}$. Suppose to the contrary that $\mathbf{p}^{(t)} \cdot \mathbf{x}^{(t)} < b^{(t)}$. Since $\mathbf{c} \cdot \mathbf{x}^* \geq \mathbf{c} \cdot \mathbf{x}^{(t)}$, then for any point $\mathbf{y} \in \operatorname{Conv}(\mathbf{x}^{(t)}, \mathbf{x}^*)$,

$$\mathbf{c} \cdot \mathbf{y} = \mathbf{c} \cdot (\alpha \mathbf{x}^{(t)} + (1 - \alpha)\mathbf{x}^*) = \alpha(\mathbf{c} \cdot \mathbf{x}^{(t)}) + (1 - \alpha)(\mathbf{c} \cdot \mathbf{x}^*) \geq \mathbf{c} \cdot \mathbf{x}^{(t)} \qquad \forall \alpha \in [0, 1].$$

Since $\mathbf{x}^{(t)}$ strictly satisfies the new constraint, there exists some point $\mathbf{y}^* \in \operatorname{Conv}(\mathbf{x}^{(t)}, \mathbf{x}^*)$ where $\mathbf{y}^* \neq \mathbf{x}^{(t)}$ such that $\mathbf{y}^* \in \mathcal{P}^{(t)}$ (i.e. $\mathbf{y}^*$ is also feasible). It follows that $\mathbf{c} \cdot \mathbf{y}^* \geq \mathbf{c} \cdot \mathbf{x}^{(t)}$, which contradicts Assumption 1. Therefore, $\mathbf{x}^{(t)}$ must bind the additional constraint. Furthermore, by non-degeneracy Assumption 5, $\mathbf{x}^{(t)}$ binds exactly $(d - 1)$ constraints in $\mathcal{P}$, i.e. $\mathbf{x}^{(t)}$ lies at the intersection of $d - 1$ hyperplanes of $\mathcal{P}$ which are linearly independent by Assumption 4. Therefore, $\mathbf{x}^{(t)}$ must be on an edge of $\mathcal{P}$.

$\square$

Proof of Lemma 3. Without loss of generality, let us assume $\mathbf{y}$ can be written as convex combination of $\mathbf{x}$ and $\mathbf{z}$ i.e. $\mathbf{y} = \alpha \mathbf{x} + (1 - \alpha)\mathbf{z}$ for some $\alpha \in (0, 1)$. Let $B_y = \{j \mid A_j \mathbf{y} = \mathbf{b}_j\}$ be the set of binding constraints for $\mathbf{y}$. We know that $|B_y| \geq d - 1$ by Assumption 5. For any j in B_y, we consider the following two cases.

1. At least one of $\mathbf{x}$ and $\mathbf{z}$ belongs to the hyperplane $\{\mathbf{w} \mid A_j \mathbf{w} = \mathbf{b}_j\}$. Then we claim that all three points bind the same constraint. Assume that $A_j \mathbf{x} = \mathbf{b}_j$, then we must have

$$A_j \mathbf{z} = \frac{A_j(\mathbf{y} - \alpha \mathbf{x})}{(1 - \alpha)} = \frac{\mathbf{b}_j - \alpha \mathbf{b}_j}{(1 - \alpha)} = \mathbf{b}_j.$$

Similarly, if we assume $A_j \mathbf{z} = \mathbf{b}_j$, we will also have $A_j \mathbf{x} = \mathbf{b}_j$.

2. None of $\mathbf{x}$ and $\mathbf{z}$ belongs to the hyperplane $\{\mathbf{w} \mid A_j \mathbf{w} = \mathbf{b}_j\}$ i.e. $A_j \mathbf{x} < \mathbf{b}_j$ and $A_j \mathbf{z} < \mathbf{b}_j$ both hold. Then we can write

$$\mathbf{b}_j = A_j \mathbf{y} = \alpha A_j \mathbf{x} + (1 - \alpha)A_j \mathbf{z} < \alpha \mathbf{b}_j + (1 - \alpha)\mathbf{b}_j = \mathbf{b}_j,$$

which is a contradiction.

It follows that for any $j \in B_y$, we have $A_j \mathbf{x} = A_j \mathbf{y} = A_j \mathbf{z} = \mathbf{b}_j$. Since $|B_y| \geq d - 1$, we know by Assumption 4 that the set of points that bind any set of $d - 1$ constraints in B_y will form an edge-space and further this edge-space will include $\mathbf{x}, \mathbf{y}$, and $\mathbf{z}$. $\square$

Proof of Lemma 4. First, note that the observed solution $\mathbf{x}^{(t)}$ is a vertex in the polytope $\mathcal{P}^{(t)} = \mathcal{P} \cap \widetilde{\mathcal{N}}^{(t)}$, that is an intersection of *exactly* d constraints by Assumption 1 and Assumption 5. Second, note that all points in e bind at least $d - 1$ constraints in $\mathcal{P}$ and since $e \subseteq \widetilde{\mathcal{N}}^{(t)}$, then all points in e bind at least d constraints in $\mathcal{P}^{(t)}$. It follows that any vertex of $\mathcal{P}^{(t)}$ on e must bind at least $(d + 1)$ constraints, which rules out the possibility of $\mathbf{x}^{(t)}$ being on e. $\square$

B.2 Section 3.2

Proof of Lemma 5. As soon as `LearnEdge` invokes update rule **U.1**, it records the solution $\mathbf{x}^* \equiv \arg\max_{\mathbf{x}\in\mathcal{P}} \mathbf{c}\cdot\mathbf{x}$. Then, the prediction rule specified by **P.1** prevents further updates of this type. This is because $\mathbf{x}^*$ continues to remain optimal if it feasible in the more constrained problem (optimizing over the polytope $\mathcal{P}^{(t)}$). $\hfill\square$

Proof of Lemma 6. Rule **U.2** is invoked only when $\mathbf{x}^{(t)} \notin X^{(t)}$ and $\mathbf{x}^{(t)} \notin e$ for any of $e \in E^{(t)}$. So after each invokation, a new point on the edge of $\mathcal{P}$ is observed. Whenever 3 points are observed on the same edge of $\mathcal{P}$, the edge-space is learned by Lemma 3 (since the points are necessarily collinear). Hence, the total number of times rule **U.2** can be invoked is at most $3|E_{\mathcal{P}}|$. $\hfill\square$

We now introduce Lemmas 16 and 17 that will be used in the proof of Lemma 18 which itself will be useful in the proof of Lemma 7. But first, for completeness, in Lemma 15 we show that we are guaranteed the existence of an edge-space if the update implemented is **U.3** or **U.4**.

Lemma 15.

*(1) If update rule **U.3** is used, then there exists edge-space $\hat{e} \in E^{(t)}$ such that $\hat{\mathbf{x}}^{(t)} \in \hat{e}$.*

*(2) If update rule **U.4** is used, then there exists edge-space $e \in E^{(t)}$ such that $\mathbf{x}^{(t)} \in e$.*

Proof. We prove this by contradiction. First consider the case in which $\mathbf{c}\cdot\hat{\mathbf{x}}^{(t)} > \mathbf{c}\cdot\mathbf{x}^{(t)}$ and suppose $\hat{\mathbf{x}}^{(t)} \in \{\mathbf{x} \in X^{(t)} \mid \forall e \in E^{(t)}, \mathbf{x} \notin e\}$. When this is the case we know that $\hat{\mathbf{x}}^{(t)}$ is feasible at day t and this contradicts $\mathbf{x}^{(t)}$ being optimal at that day because $\mathbf{c}\cdot\hat{\mathbf{x}}^{(t)} > \mathbf{c}\cdot\mathbf{x}^{(t)}$.

Next consider the case in which $\mathbf{c}\cdot\hat{\mathbf{x}}^{(t)} < \mathbf{c}\cdot\mathbf{x}^{(t)}$ and suppose $\mathbf{x}^{(t)} \in \{\mathbf{x} \in X^{(t)} \mid \forall e \in E^{(t)}, \mathbf{x} \notin e\}$. We would have used **P.3** to make a prediction because $\widetilde{\mathcal{N}}^{(t)} \cap \mathrm{Ext}^{(t)}$ is non-empty and includes at least the point $\mathbf{x}^{(t)}$. Note that by **P.3**, we have $\hat{\mathbf{x}}^{(t)} = \arg\max_{\widetilde{\mathcal{N}}^{(t)}\cap\mathrm{Ext}^{(t)}} \mathbf{c}\cdot\mathbf{x}$. Since $\mathbf{x}^{(t)} \in \widetilde{\mathcal{N}}^{(t)} \cap \mathrm{Ext}^{(t)}$, we must also have $\mathbf{c}\cdot\hat{\mathbf{x}}^{(t)} \geq \mathbf{c}\cdot\mathbf{x}^{(t)}$, which is again a contradiction. $\hfill\square$

Lemma 16. *If **U.3** is implemented at day t, then $\hat{\mathbf{x}}^{(t)} \notin \mathcal{P}$ and $\hat{\mathbf{x}}^{(t)} \in (Q_{\hat{e}}^i)^{(t)} \cap Ext^{(t)}$ for some $i = 0$ or 1 where $\hat{e}$ is given in **U.3**.*

Proof. Each time the algorithm makes update **U.3** we know that the algorithm's prediction $\hat{\mathbf{x}}^{(t)}$ was on some edge-space $\hat{e} \in E^{(t)}$ by Lemma 15. Therefore, `LearnEdge` did not use **P.1** or **P.2** to predict $\hat{\mathbf{x}}^{(t)}$. So we only need to check **P.3** and **P.4**.

- If **P.3** was used, we know that $\hat{\mathbf{x}}^{(t)} \in \widetilde{\mathcal{N}}^{(t)}$ but $\hat{\mathbf{x}}^{(t)}$ must violate a constraint of $\mathcal{P}$, due to $\mathbf{x}^{(t)}$ being the observed solution and having lower objective value. This implies that $\hat{\mathbf{x}}^{(t)}$ is in some questionable region, say $(Q_{\hat{e}}^i)^{(t)}$ for $i = 0$ or 1 but also in the extended feasible on $\hat{e}$, i.e. $\hat{\mathbf{x}}^{(t)} \in \mathrm{Ext}^{(t)} \cap (Q_{\hat{e}}^i)^{(t)}$.

- If **P.4** was used, then $\mathrm{Ext}^{(t)} = \emptyset$. However `LearnEdge` selected $\hat{\mathbf{x}}^{(t)}$ from $\mathrm{Cand}^{(t)} \neq \emptyset$ with the lowest objective value. Finally, when updating with **U.3** *(i)* $\mathbf{x}^{(t)} \in \mathrm{Cand}^{(t)}$ and *(ii)* $\mathbf{c}\cdot\mathbf{x}^{(t)} < \mathbf{c}\cdot\hat{\mathbf{x}}^{(t)}$. So we could not have used **P.4** to predict $\hat{\mathbf{x}}^{(t)}$.

$\hfill\square$

Lemma 17. *If **U.4** is implemented at day t, then $\mathbf{x}^{(t)} \in (Q_e^i)^{(t)} \backslash Ext^{(t)}$ for some $i = 0$ or 1 where e is given in **U.4**.*

Proof. As in Lemma 16, `LearnEdge` did not use **P.1** or **P.2** to predict $\hat{\mathbf{x}}^{(t)}$ (again by application of Lemma 15). So we only need to check **P.3** and **P.4**.

- If **P.3** was used, then `LearnEdge` did not guess $\mathbf{x}^{(t)}$ which had the higher objective because it was outside of $\mathrm{Ext}^{(t)}$ along edge-space e. Since $\mathbf{x}^{(t)}$ is feasible, it must have been on some questionable region on e, say $(Q_e^i)^{(t)}$ for some $i = 0$ or 1. Hence, $\mathbf{x}^{(t)} \in (Q_e^i)^{(t)} \backslash \mathrm{Ext}^{(t)}$.

- If **P.4** was used, then $\text{Ext}^{(t)} = \emptyset$ and thus $\mathbf{x}^{(t)}$ was a candidate solution but outside of the extended feasible interval along edge-space e. Further, because $\mathbf{x}^{(t)} \in \mathcal{P}$ we know that $\mathbf{x}^{(t)}$ must be in some questionable interval along e, say $(Q_e^i)^{(t)}$ for some $i = 0$ or 1. Therefore, $\mathbf{x}^{(t)} \in (Q_e^i)^{(t)} \backslash \text{Ext}^{(t)}$.

$\square$

Lemma 18. *Each time **U.3** or **U.4** is used, there is a questionable interval on some edge-space whose length is decreased by at least a factor of two.*

Proof. From Lemma 16 we know that if **U.3** is used then $\hat{\mathbf{x}}^{(t)} \in \hat{e}$, is infeasible but outside of the known infeasible interval $(Y_{\hat{e}}^i)^{(t)}$ and inside of the extended feasible interval along $\hat{e}$. Note that if a point $\mathbf{x}$ is infeasible along edge space $\hat{e}$ in the questionable interval $(Q_{\hat{e}}^i)^{(t)}$, then the constraint it violates is also violated by all points in Y_e^i. Hence the interval $\text{Conv}(\mathbf{x}, (Y_e^i)^{(t)})$ contains only infeasible points. By the definition of $(M_e^i)^{(t)}$ and the fact that $\hat{\mathbf{x}}^{(t)}$ is in the extended feasible region on $\hat{e}$, we know that

$$|(Q_e^i)^{(t+1)}| = \left|(Q_e^i)^{(t)} \backslash \text{Conv}\left(\hat{\mathbf{x}}^{(t)}, (Y_e^i)^{(t)}\right)\right| \leq \left|(Q_e^i)^{(t)} \backslash \text{Conv}\left((M_e^i)^{(t)}, (Y_e^i)^{(t)}\right)\right| = \frac{|(Q_e^i)^{(t)}|}{2}.$$

Further, from convexity we know that if $\mathbf{x}^{(t)}$ is feasible on edge-space e at day t, then the interval $\text{Conv}(\mathbf{x}^{(t)}, F_e^{(t)})$ only contains feasible points on e. We know that $\mathbf{x}^{(t)}$ is feasible and in a questionable interval $(Q_e^i)^{(t)}$ along edge space e but outside its extended feasible region, by Lemma 17. Thus, by definition of the midpoint $(M_e^i)^{(t)}$ we have

$$|(Q_e^i)^{(t+1)}| = \left|(Q_e^i)^{(t)} \backslash \text{Conv}\left(\hat{\mathbf{x}}^{(t)}, (F_e)^{(t)}\right)\right| \leq \left|(Q_e^i)^{(t)} \backslash \text{Conv}\left((M_e^i)^{(t)}, F_e^{(t)}\right)\right| = \frac{|(Q_e^i)^{(t)}|}{2}.$$

$\square$

Proof of Lemma 7. Let Q_e^i be the updated questionable interval. We know initially Q_e^i has length at most than $2\sqrt{d}$ by Assumption 2. In Lemma 18 we showed that each time an update **U.3** or **U.4** is invoked, the length of Q_e^i is decreases by at least a half. Then after at most $O(N \log(d))$ updates, the interval will have length less than 2^{-N} after which the interval will be updated at most once because there is at most one point up to precision N in it.

Therefore, the total number of updates on Q_e^i is bounded by $O(N \log(d))$. Since there are at most $2|E_{\mathcal{P}}|$ questionable intervals, the total number of updates **U.3** and **U.4** is bounded by $O(|E_{\mathcal{P}}|N \log(d))$. $\square$

B.3 Section 3.3

We prove the lower bound in Theorem 8 initially for $d = 3$.

Theorem 19. *If Assumptions 1 and 3 hold, then the number of mistakes of any learning algorithm in the known objective problem is at least $\Omega(N)$ for $d = 3$.*

Proof. The high level idea of the proof is as follows. In each day the adversary can pick two points on the two bold edges in Figure 2 as the optimal points and no matter what the learner predicts, the adversary can return a point that is different than the guess of the learner as the optimal point. If the adversary picks the midpoint of the questionable region in each day, then the size of the questionable region in both of the lines will shrink in half. So this process can be repeated N times where each entry of every vertex can be written with as a multiple of $1/2^N$, by Assumption 3. Finally, we show that at the end of this process, the adversary can return a simple polytope which is consistent with all the observed optimal points so far.

We formalize this high level in procedure ADVERSARY that takes as input any learning algorithm $\mathcal{L}$ and interacts with $\mathcal{L}$ for N days. Each day the adversary presents a constraint. Then no matter what $\mathcal{L}$ predicts, the adversary ensures that $\mathcal{L}$'s prediction is incorrect. After N interactions, the adversary outputs a feasible polytope that is consistent with all of the previous actions of the adversary.
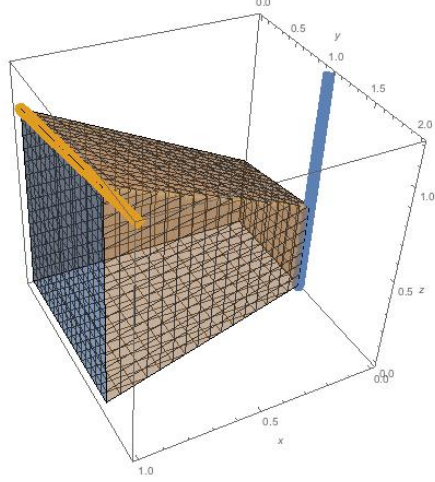
Figure 2: The underlying polytope in the proof of Theorem 8. The two learned edges are in bold.

In procedure `ADVERSARY`, subroutines `NAC` and `AD-2` are used to pick a constraint and return an optimal point that causes $\mathcal{L}$ to make a mistake, respectively. We use the notation $\mathrm{mid}(R)$ in subroutines `NAC` and `AD-2` to denote the middle point of a real interval R, $\mathrm{top}(R)$ to be the largest point in R, and $\mathrm{bot}(R)$ to be the smallest value in R. Finally, we assume the known objective function is $c = (0, 0, 1)$.

Algorithm 5 Adversary Updates (`ADVERSARY`)

Input: Any learning algorithm $\mathcal{L}$ and bit precision N
Output: Polytope $\mathcal{P}$ that is consistent with $\mathcal{L}$ making a mistake each day.
   procedure `ADVERSARY`$(\mathcal{L}, N)$
      Set $R_1^{(0)} = [0, 1], R_2^{(0)} = [1, 2]$.   ▷ Initialize
      for $t = 1, \cdots, N$ **do**
          $\left(\left(\mathbf{p}^{(t)}, q^{(t)} \right), r_1^{(t)}, r_2^{(t)} \right) \leftarrow$ `NAC`$(R_1^{(t-1)}, R_2^{(t-1)})$.
         Show constraint $\mathbf{p}^{(t)} \cdot \mathbf{x} \leq q^{(t)}$ to $\mathcal{L}$.   ▷ Constraint
         Get prediction $\hat{\mathbf{x}}^{(t)}$ from $\mathcal{L}$.
         $\left(\mathbf{x}^{(t)}, R_1^{(t)}, R_2^{(t)} \right) \leftarrow$ `AD-2` $\left(R_1^{(t-1)}, R_2^{(t-1)}, r_1^{(t)}, r_2^{(t)}, \hat{\mathbf{x}}^{(t)} \right)$.   ▷ Update
         Reveal the optimal $\mathbf{x}^{(t)} \neq \hat{\mathbf{x}}^{(t)}$ and update the regions $R_1^{(t)}$ and $R_2^{(t)}$.
      $A, \mathbf{b} \leftarrow$ `MATRIX`(R_1^N, R_2^N) ▷ Constraint matrix consistent with $\{\mathbf{x}^{(t)} \mid t \in [N]\}$
   return $A, \mathbf{b}$
   end procedure

The procedure `NAC` takes as input two real valued intervals and then outputs two points r_1 and r_2 as well as the new constraint denoted by the pair $(\mathbf{p}, q)$. The two points will be used as input in `AD-2` along with the learner's prediction. In procedure `AD-2` the adversary makes sure that the learner suffers a mistake. On each day, one of the points say r_2 produced by `NAC` has a higher objective than the other one. If the learner chooses r_2 then the adversary will simply choose a polytope that makes r_2 infeasible so that r_1 is actually the optimal point that day. If the learner chooses r_1 then the adversary picks r_2 as the optimal solution. Note that the three points r_1, r_2, and r_3 computed in `NAC` all bind the constraint and are not collinear, and thus uniquely define the hyperplane $\{\mathbf{x} : \mathbf{p} \cdot \mathbf{x} = q\}$. Finally, in `AD-2` the adversary updates the new feasible region for her use in the next days.

`ADVERSARY` finishes by actually outputting the polytope that was consistent with the constraints and the optimal solutions he showed at each day. This polytope is defined by constraint matrix A and vector $\mathbf{b}$ using the subroutine `MATRIX` as well as the nonnegativity constraint $\mathbf{x} \geq 0$.

To prove that the procedure given in `ADVERSARY` does in fact make every learner $\mathcal{L}$ make a mistake at every day, we need to show that *(i)* there exists a simple unknown polytope that is consistent with what the adversary has presented in the previous days. Furthermore, we need to show that *(ii)* the optimal point returned by the adversary on each day is indeed the optimal point corresponding to the LP with objective $\mathbf{c}$ and unknown constraints subject to the additional constraint added on each day.

15

Algorithm 6 New Adversarial Constraint (`NAC`)

---

procedure `NAC`(R_1, R_2)
    Set $\epsilon \leftarrow 0.01$.
    Set $r_1 \leftarrow (0, 1, \text{mid}(R_1))$
        $r_2 \leftarrow (1, \text{mid}(R_2), 1 + \epsilon \cdot \text{mid}(R_2))$
        $r_3 \leftarrow (1, \text{mid}(R_2), 0)$.
    Set $\mathbf{p} = (1 - \text{mid}(R_2), 1, 0)$ and $q = 1$ $\triangleright$ The constraint is $\mathbf{p} \cdot \mathbf{x} \le 1$ and binds at r_1, r_2, r_3
    return $(\mathbf{p}, q)$ and r_1, r_2.
end procedure

Algorithm 7 Adaptive Adversary (`AD-2`)

---

procedure `AD-2`$(R_1, R_2, r_1, r_2, \hat{\mathbf{x}})$
    if $\hat{\mathbf{x}} == r_2$ **then** $\triangleright$ r_1 and r_2 as in Algorithm 6
        $\mathbf{x} \leftarrow r_1$.
        $R_2 \leftarrow [\text{bot}(R_2), \text{mid}(R_2)]$.
    else
        $\mathbf{x} = r_2$.
        $R_2 \leftarrow [\text{mid}(R_2), \text{top}(R_2)]$.
    $R_1 \leftarrow [\text{mid}(R_1), \text{top}(R_1)]$.
    return $\mathbf{x}, R_1, R_2$
end procedure

To show *(i)* note that point $r_1^{(t)} = (0, 1, \text{mid}(R_1^{(t-1)}))$ is always a feasible point for $t \in [N]$ in the polytope given by A and $\mathbf{b}$ and the new constraint added each day will allow $r_1^{(t)}$ to remain feasible.

To show *(ii)* first note that the new constraint added is always a binding constraint. So by Assumption 5, it is sufficient to check the intersection of the edges of the polytope output by `MATRIX` and the newly added hyperplane and return the (feasible) point with the highest objective as the optimal point. Second, the following equations define the edges of the polytope which are one dimensional subspaces $\mathbf{e}^{i,j}$ according to Assumption 4 with A and $\mathbf{b}$ being the output of `MATRIX`.

$$\mathbf{e}^{i,j} = \{\mathbf{x} \in \mathbb{R}^3 \mid A_i \mathbf{x} = b_i \quad \text{and} \quad A_j \mathbf{x} = b_j\} \qquad i, j \in \{1, 2, 3, 4\}, \quad i \ne j,$$

where A_i is the ith row of A. Since the first two constraints define two parallel hyperplanes, we only need to consider 5 edges. Let

$$r_1^{(t)} = \left(0, 1, \text{mid}(R_1^{(t-1)})\right),$$

and

$$r_2^{(t)} = \left(1, \text{mid}(R_2^{(t-1)}), 1 + \epsilon \cdot \text{mid}(R_2^{(t-1)})\right).$$

We show that the new constraint either intersects the edges of the polytope at $r_1^{(t)}$ or $r_2^{(t)}$ or do not intersect with them at all. This will prove that the optimal points shown by the adversary each day is consistent with the unknown polytope.

    1. $\mathbf{e}^{1,4} = (0, 0, f_1) \cdot s + (0, 1, 0)$ that intersects with the new hyperplane at $r_1^{(t)}$.

Algorithm 8 Matrix consistent with adversary (`MATRIX`)

---

procedure `MATRIX`(R_1, R_2)
    Set $f_1 = (\text{top}(R_1) + \text{bot}(R_1))/2$ and $f_2 = (\text{top}(R_2) + \text{bot}(R_2))/2$ and $\epsilon > 0$

$$A \leftarrow \begin{pmatrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ (f_1 - 1 - \epsilon) & -\epsilon & 1 \\ -(f_2 - 1) \cdot f_1 & f_1 & 0 \end{pmatrix} \text{ and } \mathbf{b} \leftarrow \begin{pmatrix} 0 \\ 1 \\ f_1 - \epsilon \\ f_1 \end{pmatrix}.$$

    return A and $\mathbf{b}$
end procedure

---

2. $\mathbf{e}^{2,3} = (0, f_2, \epsilon \cdot f_2) \cdot s + (1, 0, 1)$ that intersects with the new hyperplane at $r_2^{(t)}$.

3. $\mathbf{e}^{2,4} = (0, 0, 1 + \epsilon \cdot f_2) \cdot s + (1, f_2, 0)$ that does not intersect the new hyperplane unless $\mathrm{mid}(R_2) = f_2$ (which does not happen).

4. $\mathbf{e}^{3,4} = (1, f_2 - 1, 1 + \epsilon \cdot f_2 - f_1) \cdot s + (0, 1, f_1)$ that does not intersect the new hyperplane unless $\mathrm{mid}(R_2) = f_2$ (which does not happen).

5. $\mathbf{e}^{1,3} = (0, -1, f_1) \cdot s + (0, 1, f_1)$ never intersects the hyperplane.

And this concludes the proof. □

We now prove Theorem 8 even for $d > 3$.

Proof of Theorem 8. We modify the proof of Theorem 19 to $d > 3$ by adding dummy variables. These dummy variables are denoted by $x_{4:d}$. Furthermore, we add dummy constraints $x_i \geq 0$ for all the dummy variables. We modify the objective function in the proof of Theorem 19 to be $\mathbf{c} = (0, 0, 1, -1, \ldots, -1)$. This will cause all the newly added variables to have no effect on the optimization (they should be set to 0 in the optimal solution) and, hence, the result from Theorem 19 extends to the case when $d > 3$. □

B.4 Section 3.4

Proof of Lemma 10. First, since all of the points $x_1, \ldots, x_t$ are drawn i.i.d. from $\mathcal{D}$, we observe by symmetry that the event we are interested in is distributed identically to the following event: draw a set of t points $X' = \{x_1, \ldots, x_t\}$ i.i.d. from $\mathcal{D}$ and select an index $i \in \{1, \ldots, t\}$ uniformly at random and compute the probability that $x_i \notin \mathrm{Conv}(X' \setminus \{x_i\})$. In other words

$$\Pr_{x_1, \ldots, x_t \sim \mathcal{D}} [x_t \notin \mathrm{Conv}(X)] = \Pr_{x_1, \ldots, x_t \sim \mathcal{D}, i \sim \{1, \ldots, t\}} [x_i \notin \mathrm{Conv}(X' \setminus \{x_i\})]. \tag{9}$$

We analyze the quantity on the right hand side of (9) instead, fixing the choices of $x_1, \ldots, x_t$, and analyzing the probability only over the randomness of the choice of index i. For each edge $e \in E_{\mathcal{P}}$, let $X'_e = X' \cap e$. Since each edge lies on a one dimensional subspace, there are at most two extreme points $x_1^e, x_2^e \in X'_e$ that lie outside of the convex hull of other points i.e. such that $x_1^e \notin \mathrm{Conv}(X' \setminus \{x_1^e\})$ and $x_2^e \notin \mathrm{Conv}(X' \setminus \{x_2^e\})$. We note that when we choose an index i uniformly at random, the probability that we select a point $x \in X'_e$ is exactly $|X'_e|/t$, and conditioned on selecting a point $x \in X'_e$, the probability that x is an extreme point (i.e. $x \in \{x_1^e, x_2^e\}$) is at most $2/|X'_e|$. Hence, we can calculate

$$\Pr[x_i \notin \mathrm{Conv}(X' \setminus \{x_i\})] = \sum_{e \in E_{\mathcal{P}}} \Pr[x_i \in X'_e] \cdot \Pr[x_i \notin \mathrm{Conv}(X'_e \setminus \{x_i\}) \mid x_i \in X'_e]$$

$$\leq \sum_{e \in E_{\mathcal{P}}} \frac{|X'_e|}{t} \cdot \frac{2}{|X'_e|} = \sum_{e \in E_{\mathcal{P}}} \frac{2}{t} = \frac{2|E_{\mathcal{P}}|}{t}.$$

□

Proof of Theorem 9. First, we show that `LearnHull` makes a mistake only if the true optimal point $\mathbf{x}^{(t)}$ lies outside of the convex hull $\mathcal{C}^{(t-1)}$ formed by the previous observed optimal points $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t-1)}\}$. Suppose that at day t, the algorithm predicts the point $\hat{\mathbf{x}}^{(t)}$ instead of the optimal point $\mathbf{x}^{(t)}$. Since each point in $\mathcal{C}^{(t-1)}$ is feasible and $\hat{\mathbf{x}}^{(t)}$ is the point with the highest objective value among the points in $\{\mathbf{x} \in \mathcal{C}^{(t-1)} \mid \mathbf{p}' \cdot \mathbf{x} \leq b'\}$, then it must be that $\mathbf{x}^{(t)} \notin \mathcal{C}^{(t-1)}$ because otherwise $\mathbf{c} \cdot \mathbf{x}^{(t)} > \mathbf{c} \cdot \hat{\mathbf{x}}^{(t)}$. By Lemma 10, we also know that the probability that $\mathbf{x}^{(t)}$ lies outside of $\mathcal{C}^{(t-1)}$ is no more than $2|E_{\mathcal{P}}|/t$ in expectation, which also upper bounds the probability of `LearnHull` making a mistake at day t. Therefore, the expected number of mistakes made by `LearnHull` over T days is bounded by the sum of probabilities of making a mistake in each day which is $\sum_{t=1}^{T} 2|E_{\mathcal{P}}|/t = O(|E_{\mathcal{P}}| \log(T))$. □

Proof of Theorem 11. The deterministic procedure runs $\lceil 18 \log(1/\delta) \rceil$ independent instances of the `LearnHull` each using independently drawn examples. The independent instances are aggregated into a single prediction rule by predicting using the *modal* prediction (if one exists), and otherwise

17

predicting arbitrarily. Hence, the aggregate prediction is correct whenever at least half of the instances of `LearnHull` are correct.

We show that if each instance of the `LearnHull` is run for $8|E_\mathcal{P}|$ days, then the probability that more than half of the instances of `LearnHull` make a mistake on a newly drawn constraint at day $T + 1$ is at most δ. The result is that with probability at least $1 - \delta$ the majority of instances of `LearnHull` predict the correct optimal point, and hence the aggregate prediction is also correct.

Let Z_i be the random variable that denotes the probability that the ith instance of the `LearnHull` algorithm makes a mistake on a fresh example, after it has been trained for $8|E_\mathcal{P}|$ days. By Theorem 9, we know $E[Z_i] \leq 1/4$ for all i. Now by Markov's inequality,

$$\Pr\left[Z_i \geq \frac{3}{4}\right] = \Pr\left[Z_i \geq 3 \cdot E[Z_i]\right] \leq 1/3,$$

for all i. Hence, the *expected* number of instances that make a mistake is at most $1/3$. Finally, since each instance is trained on independent examples, a Chernoff bound implies that the probability that at least half of the instances of `LearnHull` make a mistake is bounded by δ.

$\square$

C Circumventing the Lower Bound when $d \leq 2$

In Theorem 8 (in Section 3.3), we proved the necessity of Assumption 3 by showing that the dependence on the precision parameter N in our mistake bound is tight. However, Theorem 8 requires the dimension d to be at least 3.

We now show that this condition on the dimension is indeed necessary—even without the finite precision assumption (Assumption 3), we can have (computationally efficient) algorithms with small mistake bounds when the dimension $d \leq 2$.

In $d = 1$, at most two constraints are sufficient to determine any constraint matrix A because the constraint matrix A defines a feasible interval on the real line. So we will guess the value that maximizes the objective subject to the single known constraint. Once we have made a mistake, we must have learned the true optimal to the underlying problem because our guess was infeasible. After this single mistake, we either guess the true optimal that we have already seen or if it is not feasible with the new constraint then we guess the point that maximizes the objective subject to the new constraint. Thus after one mistake, the learner will not make any more mistakes.

Lemma 3 tells us that the line between any three collinear points must give us an edge-space of the underlying polytope. When $d = 2$, the corresponding edge-space is then just one of the original constraints of the underlying polytope. Since each solution must be on an edge of the underlying polytope each day, we can make at most $3m$ mistakes without seeing the true objective. Hence, all together, we can make at most $3m + 1$ mistakes before we recover all the constraints of the underlying polytope, or all the rows of the constraint matrix A, and see the true optimal solution.

This phenomenon does not continue to hold for $d > 2$ (as we show in our lower bound in Theorem 8).

D Missing Proofs from Section 4

First we state Theorem 20 from Grotschel et al. [11] about the running time of the Ellipsoid algorithm.

Theorem 20. *Let $\mathcal{P} \subset \mathbb{R}^d$ be a polytope given as the intersection of linear constraints, each specified with N bits of precision. Given access to a separation oracle which can return, for each candidate solution $p \notin \mathcal{P}$ a hyperplane with N bits of precision that separates p from $\mathcal{P}$, the Ellipsoid algorithm outputs a point $p' \in \mathcal{P}$ or outputs $\mathcal{P}$ is empty at most $\mathrm{poly}(d, N)$ iterations.*

We are now ready to bound the number of mistakes that `LearnEllipsoid` makes.

Proof of Theorem 13. Whenever `LearnEllipsoid` makes a mistake, there exists a separating hyperplane

$$\mathcal{H}^{(t)} = \left\{ W = (\mathbf{w}^1, \ldots, \mathbf{w}^n) \in \mathbb{R}^{n \times d} \mid \sum_{i \in S^{(t)}} \mathbf{w}^i \cdot (\mathbf{x}^{(t)} - \hat{\mathbf{x}}^{(t)}) > 0 \right\}$$

that cause the Ellipsoid algorithm to run for another iteration. When `LearnEllipsoid` reduces $\mathcal{F}$ to a set that contains a single point up to N bits of precision then predicting via the above equation for $\mathcal{H}^{(t)}$ will ensure we never make a mistake again. Hence, the number of mistakes that `LearnEllipsoid` can commit against an adversary is bounded by the maximum number of iterations for which the Ellipsoid algorithm can be made to run, in the worst case.

We know that $\mathbf{x}^{(t)}$ each day is on a vertex of the polytope $\mathcal{P}^{(t)}$ which is guaranteed to have coordinates specified with at most N bits of precision by Assumption 7. Theorem 12 guarantees that the solution $\hat{\mathbf{x}}^{(t)}$ that `LearnEllipsoid` produces using the following equation

$$\hat{\mathbf{x}}^{(t)} \in \operatorname*{argmax}_{\mathbf{x} \in \mathcal{P}^{(t)}} \left\{ \sum_{i \in S^{(t)}} (\mathbf{w}^i)^{(t)} \cdot \mathbf{x} \right\}$$

is a vertex solution of $\mathcal{P}^{(t)}$. So $\hat{\mathbf{x}}^{(t)}$ can also be written with N bits of precision by Assumption 7. Thus, every constraint in $\mathcal{F}$ and hence each separating hyperplane can be written with $d \cdot N$ bits of precision. By Assumption 6 and Theorem 20, we know that the Ellipsoid algorithm will find a point in the feasible region $\mathcal{F}$ after at most $\operatorname{poly}(n, d, N)$ many iterations which is the same as the number of mistakes of `LearnEllipsoid`. $\qquad\square$